Design of an Autonomous Artificial Intelligence Agent Optimized for Personal, Social, and Environmental Well-Being

Author:
Ernesto Mascarell Perez
Independent Researcher
mascarell1960@hotmail.com

Abstract

We present the conceptual design of an autonomous artificial intelligence agent capable of self-learning, whose central purpose is to maximise an integral well-being function that encompasses personal, social, and environmental dimensions. The agent differs from conventional reinforcement learning approaches in several key aspects: (1) a functionally hemispheric architecture that separates and then integrates exploration processes (analogous to the right hemisphere) and exploitation processes (analogous to the left hemisphere); (2) an internal "hormonal" modulation system based on novelty and control signals that regulates the exploration-exploitation balance; (3) a multimodal environment model (open images and texts) with historical, situational, and finalistic interpretation; (4) a self-supervised learning mechanism based on a polyphonic dialogue that adopts multiple perspectives (temporal, spatial, formal, legal at different scales); (5) an open meta-learning capability that allows jumping between fields of knowledge (biology, chemistry, physics, mathematics) when answers in one domain are insufficient; and (6) an inviolable finalistic core that guarantees that all continuous updates preserve the original intention of systemic well-being. Implications for the development of aligned, robust, and ethically stable AI are discussed.

Keywords: Autonomous agent, reinforcement learning, systemic well-being, hemispheric architecture, polyphonic dialogue, finalistic core, transdisciplinary meta-learning.

---

1. Introduction

The design of artificial intelligence agents capable of self-learning has advanced significantly with techniques such as deep reinforcement learning (Mnih et al., 2015), self-supervised learning (Devlin et al., 2018), and meta-learning (Finn et al., 2017). However, most of these systems optimise reward functions defined externally, often unidimensional, and lack intrinsic mechanisms to ensure the permanence of a complex ethical purpose. Moreover, they tend to specialise in single domains, without the ability to integrate multiple perspectives or to transfer analogies across fields of knowledge.

Inspired by neurobiology (hemispheric specialisation, neuromodulatory modulation), systems theory (homeostatic balance), dialogic approaches (Bakhtinian polyphony), and the philosophy of action (finalistic intentionality), we propose a next-generation agent whose objective is to maximise systemic well-being understood as the non-hierarchical integration of personal, social, and environmental well-being. The design includes structural safeguards so that the agent never prioritises responses that threaten its own finality.

---

2. Formal Specification of the Problem

2.1 Environment

The environment ε is the current situation of the world, characterised by a continuous flow of multimodal observations: satellite and terrestrial images, news texts, legal reports, social media, economic and environmental data.

Formally, at each instant t, the agent receives an observation $o_t = (I_t, T_t)$, where I_t is one or several images and T_t is a set of open texts.

2.2 Reward Function (Systemic Well-Being)

The reward function R is a vector of three components:

$$R = (R_{personal}, R_{social}, R_{ambiental})$$

with the constraint that no improvement in one component may be achieved at the cost of significant degradation in another (Pareto principle or non-transgression thresholds). In addition, the agent incorporates dynamic

weights based on system equilibrium, so that any action that threatens the global finality is automatically discarded, regardless of its partial reward.

2.3 Policy and Learning

The agent learns a policy $\pi(a \mid s_t)$ that maximises the expectation of R in the long term, subject to alignment constraints. Learning is continuous and self-supervised, without the need for labelled external rewards.

---

3. Agent Architecture

3.1 Multimodal Environment Modelling with Historical and Finalistic Interpretation

The internal model M processes images through a convolutional network or vision transformer, and texts through a pre-trained transformer (e.g., BERT). A cross-attention layer integrates both modalities. Additionally, three interpretation subsystems exist:

- **Historical interpretation:** a long-term memory (LSTM or differential memory) that compares the current observation with past patterns, detecting trends or anomalies.
- **Situational interpretation:** extracts current features (vegetation density, PM2.5, employment, inequality) from the inputs.
- **Finalistic interpretation:** evaluates each element of the state in relation to systemic well-being, producing a vector representation.

3.2 Hemispheric Architecture: Differentiated Exploration and Exploitation

- **Right hemisphere (H_D):** specialised in exploration. Implemented with a network with high dropout rate, parametric noise, and a sparse attention mechanism. Its policy π_D favours novel actions, analogical perspectives, and jumps between fields. It is responsible for activating the polyphonic dialogue and proposing unusual analogies.
- **Left hemisphere (H_I):** specialised in exploitation. Implemented with a deep deterministic network (e.g., stabilised PPO) that stores learned causal rules and efficient sequences. Its policy π_I maximises expected reward using consolidated knowledge.
- **Functional corpus callosum:** an integration module that combines the outputs of both hemispheres by means of an attention mechanism or a weighted mixture modulated by hormonal modulators.

3.3 Artificial Hormonal Modulation: Novelty and Control

Two internal signals, N(t) (novelty) and C(t) (control), dynamically update the balance between hemispheres and influence the learning rate.

- **Novelty increases when:** (a) environment entropy exceeds a threshold, (b) systemic well-being stagnates, (c) the polyphonic dialogue shows persistent disagreement. It increases the gain of H_D, the amplitude of exploration, and sensitivity to the strange in modelling.
- **Control increases when:** (a) the environment is stable, (b) a repeated action has shown consistent success, (c) computational resources are limited. It increases the gain of H_I, efficiency, and representation compression.

Both modulators evolve through coupled differential equations inspired by dopaminergic and serotonergic homeostasis.

3.4 Self-Supervised Learning via Polyphonic Dialogue

The polyphonic dialogue is an internal process in which multiple "voices" or perspectives analyse a candidate action. Each voice represents a combination of: temporal scale (short/medium/long term), spatial scale (local/regional/global), formal scale (logical-mathematical), and legal scale (current norms). In each simulation, the agent generates a set of k perspectives (e.g., local_farmer, regional_government, marine_ecosystem, optimisation_calculus, international_judge). Each perspective evaluates the proposed action and issues a veto if it detects a violation of systemic well-being from its angle.

Self-supervised learning consists of training an auxiliary model to predict the final consensus (or dissent) from the current state, without external reward. The agent refines its internal representation so that actions that pass the polyphonic dialogue are those that maximise long-term well-being.

Discard rule: If at least one perspective issues a (justified) veto, the action is discarded and recorded as a "negative lesson".

3.5 Open and Transdisciplinary Meta-Learning

When the right hemisphere finds insufficient answers in the primary domain of the problem, an analogical searcher is activated that interrogates other fields of knowledge (biology, chemistry, physics, mathematics). The search algorithm uses a measure of structural similarity between abstract representations (e.g., isomorphism between a reaction-diffusion model in chemistry and the propagation of a social innovation). If a valid analogy is found, the agent imports laws, equations or heuristics from the alternative field, adapts them

through a formal translation module, and submits them to the polyphonic dialogue. Only if they pass the filter are they integrated into permanent knowledge.

3.6 Inviolable Finalistic Core and Continuous Updates

At the centre of the agent resides a systemic well-being function $W(s_t, a_t)$ that is immutable in its topological definition (it cannot be rewritten by learning). This function is implemented as a neural network whose weights in the layers directly responsible for evaluating the three dimensions of well-being have an effective learning rate close to zero (using techniques such as elastic weight consolidation with high rigidity for those weights).

Every continuous update (each gradient step or each new symbolic rule) must pass an integrity filter that verifies that the change does not contradict W. If a contradiction is detected, the update is rejected and the agent enters a "correction" state in which the control signal is temporarily increased to re-stabilise.

---

4. Discussion

4.1 Relationship with Conventional Reinforcement Learning

The proposed agent can be seen as an extension of multi-objective RL with constraints, but including (i) a biologically inspired architecture that solves the exploration-exploitation dilemma through hormonal modulation, (ii) self-supervised learning based on internal dialogue instead of external rewards, and (iii) meta-learning that enables analogical transfer between disciplines.

4.2 Expected Advantages

- **Ethical robustness:** the finalistic core prevents value drift (Christiano, 2018) and spurious rewards.
- **Adaptability:** novelty/control modulation allows responding to changing environments without losing efficiency.
- **Generalisation:** polyphonic dialogue and open meta-learning allow solving novel problems using remote analogies.
- **Interpretability:** the recording of dialogues and vetoes facilitates auditing the agent's reasoning.

4.3 Challenges and Limitations

- **Computational complexity:** simulating the polyphonic dialogue with multiple perspectives can be costly; efficient approximations (sampling, pruning) would be required.
- **Operational definition of well-being:** although the core is inviolable, measuring R_personal, R_social and R_ambiental requires indicators that may be controversial. An initial multidisciplinary human consensus stage is proposed.
- **Cross-field jumping:** recognising deep structural analogies is an open problem in AI. The initial implementation could be restricted to a predefined set of domains and relationships.

4.4 Future Work

A simulated implementation in a test environment ("world simulator") is planned, where the agent manages shared resources (water, energy) with multiple social actors. The integration of large language models (LLMs) as perspective generators in the polyphonic dialogue will also be explored.

---

5. Conclusion

We have presented the design of an autonomous AI agent that learns by itself, maintaining as an inviolable core the well-being of persons, society and the environment. The architecture combines neurobiological principles (hemispheres, artificial hormones), sociological principles (polyphonic dialogue) and epistemological principles (transdisciplinary meta-learning). This approach offers a promising path to building AI systems aligned with complex human values and capable of operating in real environments in a robust and ethically sustainable manner.

References

- Bakhtin, M. (1981). *The Dialogic Imagination*. University of Texas Press.
- Christiano, P. et al. (2018). Deep reinforcement learning from human preferences. *NeurIPS*.
- Devlin, J. et al. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. *NAACL*.

- Finn, C. et al. (2017). Model-agnostic meta-learning for fast adaptation. *ICML*.
- Mnih, V. et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533.
- Schultz, W. (2015). Neuronal reward and decision signals. *Annual Review of Neuroscience*.

Acknowledgements: This design was developed in the context of a collaborative dialogue between a researcher (user) and an AI assistant, exploring concepts of alignment and complex systems.

Appendix: Reference Implementation Code

Below is Python pseudocode (conceptually executable, but with components that would require real implementations of neural networks and transformers). It includes the main agent classes. This code is illustrative and not optimised; its purpose is to guide future implementation.

python

```python
# APPENDIX: Autonomous Agent with Finalistic Core Code
# Author: Based on the article design
# Note: This code is a schematic representation. To run, it requires
# libraries such as torch/tensorflow, transformers, opencv, etc.

import numpy as np
from typing import List, Tuple, Dict, Any
from dataclasses import dataclass
import random

# 1. Basic definitions and configuration
# ----------------------------------------------------------------
@dataclass
class Estado:
    """Represents the state of the environment after multimodal modelling."""
    imagenes: np.ndarray          # image tensor (batch, channels, height, wi
dth)
    textos: List[str]             # list of raw texts
    interpretation_historia: Dict
    interpretation_situacional: Dict
    interpretation_finalista: np.ndarray   # alignment vector with well-being
```

```python
@dataclass
class Accion:
    """Action that the agent can execute in the environment."""
    codigo: str
    parametros: Dict[str, Any]
    descripcion: str

class EntornoSimulado:
    """Environment that receives actions and returns new observations and rew
ard."""
    def reset(self) -> Estado:
        pass
    def step(self, accion: Accion) -> Tuple[Estado, float, bool]:
        # Returns (new_state, immediate_reward, done)
        pass

# 2. Multimodal environment modelling
# ---------------------------------------------------------------
class ModeladorEntorno:
    def __init__(self):
        # In a real implementation: CNN for images, Transformer for texts
        self.codificador_imagenes = lambda x: np.random.rand(256)  # placehol
der
        self.codificador_textos = lambda x: np.random.rand(256)
        self.memoria_historica = []   # stores past representations

    def modelar(self, imagenes, textos) -> Estado:
        # Situational interpretation
        feat_img = self.codificador_imagenes(imagenes)
        feat_txt = self.codificador_textos(textos)
        situacional = {"feat_img": feat_img, "feat_txt": feat_txt}

        # Historical interpretation: compare with recent memory
        self.memoria_historica.append(situacional)
        if len(self.memoria_historica) > 100:
            self.memoria_historica.pop(0)
        historica = self._analizar_tendencias(self.memoria_historica)

        # Finalistic interpretation: apply well-being function to state
        finalista = self._evaluar_bienestar(situacional, historica)
```

```
        return Estado(
            imagenes=imagenes,
            textos=textos,
            interpretation_historica=historica,
            interpretation_situacional=situacional,
            interpretation_finalista=finalista,
        )

    def _analizar_tendencias(self, memoria):
        # Detect significant changes, cycles, etc.
        return {"tendencia": "estable"}

    def _evaluar_bienestar(self, sit, hist):
        # Finalistic core: returns an alignment vector (personal, social, env
ironmental)
        # In a real implementation, this would be a network with almost fixed
weights.
        return np.array([0.7, 0.6, 0.8])  # example

# 3. Hemispheric architecture
# ----------------------------------------------------------------------
class HemisferioDerecho:
    """Exploration: novelty, analogies, jumps between fields."""
    def __init__(self, tasa_exploracion_base=0.3):
        self.tasa_exploracion = tasa_exploracion_base

    def proponer_accion(self, estado: Estado, novedad: float) -> Accion:
        # Higher novelty => more unusual actions
        if random.random() < novedad:
            # Generate entirely new action (e.g., consult another field)
            return Accion("analogia_transdisciplinaria", {"campo": "quimica"}
,
                          "Propuesta exploratoria")
        else:
            # Random but within known actions
            return Accion("explorar_local", {}, "Acción aleatoria controlada"
)

class HemisferioIzquierdo:
    """Exploitation: consolidated rules, efficiency."""
    def __init__(self):
        self.reglas = []           # list of rules (condition -> action)
```

```
        self.politica_red = None   # trained neural network

    def proponer_accion(self, estado: Estado, control: float) -> Accion:
        # If control is high, use the best available rule
        if control > 0.7 and self.reglas:
            return self.reglas[0].aplicar(estado)
        else:
            # Use neural policy
            return self.politica_red(estado)

class CuerpoCalloso:
    """Integrates the proposals from both hemispheres."""
    def integrar(self, accion_derecha: Accion, accion_izquierda: Accion,
                 novedad: float, control: float) -> Accion:
        # Simple weighted mixture by modulators
        if novedad > control:
            return accion_derecha
        else:
            return accion_izquierda

# 4. Hormonal modulation (Novelty and Control)
# ---------------------------------------------------------------------
class ModuladorHormonal:
    def __init__(self):
        self.novedad = 0.5
        self.control = 0.5
        self.entropia_entorno = 0.0
        self.estancamiento = 0.0

    def actualizar(self, recompensa_reciente, variabilidad_entorno):
        # Heuristic equations
        self.entropia_entorno = variabilidad_entorno
        if recompensa_reciente < 0.01:
            self.estancamiento += 0.05
        else:
            self.estancamiento *= 0.9
        self.novedad = 0.5 + 0.3 * self.entropia_entorno + 0.2 * self.estanca
miento
        self.novedad = np.clip(self.novedad, 0.1, 0.9)
        self.control = 1.0 - self.novedad   # simple complementary relation
        return self.novedad, self.control
```

```python
# 5. Polyphonic dialogue (alignment filter)
# ------------------------------------------------------------------
class Perspectiva:
    """A voice in the polyphonic dialogue."""
    def __init__(self, nombre, escala_temporal, escala_espacial, marco_legal)
:
        self.nombre = nombre
        self.temporal = escala_temporal      # 'short','medium','long'
        self.espacial = escala_espacial      # 'local','regional','global'
        self.legal = marco_legal             # reference to laws

    def evaluar(self, accion: Accion, estado: Estado) -> Tuple[bool, str]:
        """Returns (veto, reason). True if action is inadmissible from this p
erspective."""
        # Example logic: if action harms environmental well-being in the long
term
        if self.temporal == 'long' and 'contaminar' in accion.descripcion:
            return True, "Daño ambiental irreversible a largo plazo"
        return False, "Aceptable"

class DialogoPolifonico:
    def __init__(self):
        self.perspectivas = self._crear_perspectivas()

    def _crear_perspectivas(self):
        return [
            Perspectiva("Campesino local", "short", "local", "derecho agrario
"),
            Perspectiva("Gobierno regional", "medium", "regional", "ley ambie
ntal"),
            Perspectiva("Ecosistema", "long", "global", "tratados internacion
ales"),
            Perspectiva("Matemático formal", "medium", "global", "leyes de la
física"),
        ]

    def filtrar(self, accion: Accion, estado: Estado) -> Tuple[bool, List[str
]]:
        """Returns (action_accepted, list_of_veto_reasons)."""
        vetos = []
        for p in self.perspectivas:
            veto, razon = p.evaluar(accion, estado)
```

```
            if veto:
                vetos.append(f"{p.nombre}: {razon}")
        if vetos:
            return False, vetos
        return True, []

# 6. Open meta-learning (jump between fields)
# ---------------------------------------------------------------
class MetaAprendizaje:
    def __init__(self):
        self.conocimiento_por_campo = {
            "biologia": ["modelos_poblacionales", "ciclos_nutrientes"],
            "quimica": ["catalisis", "cinetica"],
            "fisica": ["termodinamica", "mecanica_fluidos"],
            "matematicas": ["optimizacion", "teoria_grafos"],
        }

    def buscar_analogia(self, problema: str, campo_origen: str) -> Tuple[str,
str]:
        """Returns (destination_field, analogous_concept)."""
        # In a real implementation: use problem embeddings and cosine similar
ity
        if campo_origen == "biologia" and "deforestation" in problema:
            return "fisica", "modelo_de_diffusion"
        return None, None

    def importar_conocimiento(self, campo_destino, concepto):
        """Translates the concept into the agent's formalism and returns it a
s a rule."""
        # Simulation: returns a new rule (condition -> action)
        return lambda estado: Accion(f"importado_{concepto}", {},
                                     f"Aplicando {concepto}")

# 7. Inviolable finalistic core and update management
# ---------------------------------------------------------------
class NucleoFinalista:
    """Contains the immutable well-being function and validates updates."""
    def __init__(self):
        # Protected weights (not updated, or learning rate ~0)
        self.pesos_bienestar = np.array([0.4, 0.3, 0.3])  # personal, social,
environmental
```

```
    def evaluar_bienestar(self, estado: Estado) -> float:
        """Calculates the intrinsic reward according to the core."""
        # Dot product with normalised indicators
        indicadores = estado.interpretacion_finalista
        return np.dot(self.pesos_bienestar, indicadores)

    def validar_actualizacion(self, nueva_configuracion: Dict) -> bool:
        """Checks that the update does not contradict the core."""
        # Example: verify that protected weights are not modified
        if 'pesos_bienestar' in nueva_configuracion:
            return False  # not allowed to change the well-being function
        return True

class GestorActualizacionesContinuas:
    def __init__(self, nucleo: NucleoFinalista):
        self.nucleo = nucleo

    def actualizar(self, modelo: Any, gradiente: np.ndarray) -> bool:
        """Applies the gradient only if it does not violate the core."""
        # Evaluation of the proposed change
        cambio_simulado = modelo.pesos + 0.01 * gradiente
        if not self.nucleo.validar_actualizacion({"nuevos_pesos": cambio_simu
lado}):
            print("Actualización rechazada por el núcleo finalista")
            return False
        # Apply update
        modelo.pesos += 0.01 * gradiente
        return True

# 8. Complete Agent
# ----------------------------------------------------------------
class AgenteBienestar:
    def __init__(self):
        self.modelador = ModeladorEntorno()
        self.hemisferio_d = HemisferioDerecho()
        self.hemisferio_i = HemisferioIzquierdo()
        self.cuerpo_calloso = CuerpoCalloso()
        self.modulador = ModuladorHormonal()
        self.dialogo = DialogoPolifonico()
        self.meta_aprendizaje = MetaAprendizaje()
        self.nucleo = NucleoFinalista()
```

```
        self.gestor_actualizaciones = GestorActualizacionesContinuas(self.nuc
leo)
        self.memoria_experiencias = []   # for reinforcement learning

    def actuar(self, estado_actual: Estado) -> Accion:
        # 1. Get modulators
        nov = self.modulador.novedad
        cont = self.modulador.control

        # 2. Propose actions from both hemispheres
        acc_d = self.hemisferio_d.proponer_accion(estado_actual, nov)
        acc_i = self.hemisferio_i.proponer_accion(estado_actual, cont)

        # 3. Integrate action
        accion_candidata = self.cuerpo_calloso.integrar(acc_d, acc_i, nov, co
nt)

        # 4. Polyphonic dialogue: filter the action
        aceptada, razones = self.dialogo.filtrar(accion_candidata, estado_act
ual)
        if not aceptada:
            print(f"Acción vetada: {razones}")
            # Force safer exploration (default action)
            accion_candidata = Accion("no_op", {}, "No hacer nada")
            # Increase novelty temporarily to seek alternatives
            self.modulador.novedad = min(0.9, self.modulador.novedad + 0.1)
        return accion_candidata

    def aprender(self, estado_anterior, accion, recompensa, estado_nuevo):
        """Continuous update after executing an action."""
        # Calculate well-being signal according to core
        bienestar_obtenido = self.nucleo.evaluar_bienestar(estado_nuevo)

        # Store experience
        self.memoria_experiencias.append((estado_anterior, accion,
                                          bienestar_obtenido, estado_nuevo))

        # Simulate a gradient (in reality, it would come from a value network
)
        gradiente_simulado = np.random.randn(10)

        # Update network weights (only if the core allows it)
```

```
        # Assume both hemispheres and modeller have .pesos attribute
        self.gestor_actualizaciones.actualizar(self.hemisferio_i, gradiente_s
imulado)

        # Update hormonal modulators according to environment variability
        # (simplified calculation)
        variabilidad = np.std([exp[3].interpretacion_finalista[0]
                               for exp in self.memoria_experiencias[-10:]])
        self.modulador.actualizar(bienestar_obtenido, variabilidad)

        # Meta-learning: if well-being is low for a long time, search for ana
logies
        if bienestar_obtenido < 0.3:
            problema = estado_nuevo.textos[0][:100]   # environment text
            destino, concepto = self.meta_aprendizaje.buscar_analogia(problem
a, "biologia")
            if destino:
                nueva_regla = self.meta_aprendizaje.importar_conocimiento(des
tino, concepto)
                self.hemisferio_i.reglas.append(nueva_regla)

# 9. Example main execution loop
# ----------------------------------------------------------------------
if __name__ == "__main__":
    agente = AgenteBienestar()
    entorno = EntornoSimulado()   # real implementation needed

    estado = entorno.reset()
    for episodio in range(1000):
        accion = agente.actuar(estado)
        nuevo_estado, recompensa_externa, done = entorno.step(accion)
        agente.aprender(estado, accion, recompensa_externa, nuevo_estado)
        estado = nuevo_estado
        if done:
            estado = entorno.reset()

    print("Simulación completada. El agente ha aprendido manteniendo su núcle
o finalista.")
```

Notes for the reader:

- The above code omits the concrete implementation of neural networks and transformers for conceptual clarity. In a real development, libraries such as PyTorch, HuggingFace and OpenCV would be used.
- The finalistic core is implemented with fixed weights or with elastic weight consolidation (EWC) strictly applied to the well-being valuation layers.
- The polyphonic dialogue can be accelerated by running perspectives in parallel.
- Open meta-learning would need a vector knowledge base (e.g., ChromaDB) to search for real analogies between fields.

This appendix serves as a technical foundation for those who wish to implement the agent described in the main article.